DOCUMENT RESUME

ED 280 737 SE 047 901

AUTHOR Cobern, William W.
TITLE Programming Language Choice: A Positive albeit Ambiguous Case for BASIC Programming in Secondary Science Teaching.
PUB DATE Mar 86
NOTE 21p.
PUB TYPE Viewpoints (120) -- Guides - Non-Classroom Use (055)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Cognitive Development; *Computer Science Education; *Computer Uses in Education; Educational Technology; Instructional Improvement; Programing; *Programing Languages; Science Education; Secondary Education; *Secondary School Science; *Skill Development
IDENTIFIERS *BASIC Programing Language; *PASCAL Programing Language

ABSTRACT

With the purpose of addressing the area of language choice in computer programming, this paper specifically addresses the use of BASIC as a language for secondary level students. Perspectives are offered on: (1) the role of computers in science education; (2) the differences in quality and intent of PASCAL and BASIC; (3) the role of a non-structured programming language approach; and (4) the problem of cognitive matching. An increase in research is advocated for such areas as the roles of programming in science education, an analysis of the task structure and curriculum demand of languages such as PASCAL, and the effect of prior unstructured programming experience on the acquisition of structured programming skills. A reference list is also provided. (ML)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

Programming Language Choice: a Positive Albeit Ambiguous Case for BASIC programming in Secondary Science Teaching

ED280737

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

William W. Cobern

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Dr. William W. Cobern
Assistant Professor of Education
Austin College Sherman, Texas
March, 1986

SE 047 901

BEST COPY AVAILABLE

Abstract

Given the increasing computer exposure that teachers are experiencing and the growing number of personally owned computers it is likely that secondary science teachers will with greater frequency experiment with student programming. Researchers should be interested in examining the possible uses for computer programming in science education and the structure of languages that might be choosen, vis-a-vis the educational requirements of the learners. At this time BASIC and Pascal are the two most frequently used programming languages in secondary education. Although the use of BASIC has been severely criticized because of its unstructured nature, recent research results tend to support BASIC as the langauge better matched cognitvely with the development levels of typical secondary students.

Given that one is interested in using computer programming as an instructional tool in secondary science education, important questions arise. First, why should programming be used at all and second, which language or languages should be used? In this paper I primarily address the issue of language choice, more specifically the use of BASIC, rather than a structured language such as Pascal. The case for BASIC however is an ambiguous one since there is at this time insufficient empirical evidence to unequivically resolve the issue one way or the other.

Computers and Science Education

My first encounter with a computer occurred when I was an undergraduate. As science majors we were taught ALGOL so that we could write programs for solving various mathematical equations used in our physics and chemistry labs. Later as a high school science teacher, one fortunate enough to have access at school to a minicomputer via a remote terminal, I thought my students might enjoy learning to write computer programs that could be used to solve mole problems, acceleration problems, or to simply average measurements from control and experimental groups. I also felt that attempting to write programs for science-math problems could improve their understanding of those math relationships. Furthermore, because scientists use computers I felt that by having my high school students do the same, they would increase their own sense of "doing science." Like many students today my students did enjoy working with a computer; and it was my judgement

that they had a better understanding of our coursework because they were programming. But what I was doing was uncommon, given the rarity of computer access at the high school level.

With the advent of the affordable microcomputer in the late seventies, the use of computers in education has greatly increased. The recently released "Second National Survey of Instructional Uses of School Computers" estimates that half of all American secondary schools have 15 or more microcomputers and that 12 % of all secondary science teachers are now using microcomputers (Becker, 1986, pp. 30, 32). According to a RAND study of secondary science teachers who are considered "successful" users of computers in education, computers are most often used for drill and practice, tutoring, simulating, game playing and management purposes (Shavelson, et al, 1984).

It should come as no surprise that student programming is not listed in the RAND study. As noted by the Office of Technical Assessment (1982), a major hindrance to the increased use of modern technology in the classroom is that teachers lack the necessary technical knowledge. It is only to be expected then that teachers would first use the more simple forms of a new technology. With respect to microcomputers that means pre-programmed, commercially available software used for drill and practice, tutorials, etc. Furthermore there is less that can be done with student programming than can be done with prepared software. The result is that computer programming is largely left to the computer literacy and computer science teachers.

Nevertheless there is likely to be a change in the current patterns of programming use due to the increasing occurrence of computer literacy requirements. School districts are requiring computer literacy for high school graduation, colleges are doing the same for college graduation, and states are requiring it for teacher certification (Abernathy & Pettibone, 1984). These requirements coupled with the growing number of home computers means that virtually all of tomorrow's new science teachers will have at least been exposed to a computer programming language. It seems inevitable that these teachers are going to experiment with student programming for the same reasons I did.

If indeed that is what happens, then two basic questions must be faced by science education researchers:

1. Are there any instructional objectives for which an empirical justification can be shown for using student programming in secondary science education? For example, as opposed to other methods, can a motivational objective be better met with student programming, or does student programming increase understanding of science-related mathematical relationships?

2. Given that it is possible for a teacher to choose from several programming languages, does the choice of language for student programming matter? For example, is Pascal to be preferred to BASIC, or perhaps Modula is preferable?

To date there has not been any research concerning the first question nor much discussion. There has however been considerable discussion of the second question, though often in the broader context of computer science education. A teacher's decision to teach and use BASIC for example, is a controversial one since the use of BASIC is

vigorously discouraged by some eminent science educators and computer scientists (e.g. Bork, 1985; Papert, 1980). On the assumption that science teachers will go ahead and try student programming despite the absence of empirical justification for doing so, there is cause to examine what information is available on the issue of language choice.

The Anti-BASIC Position

BASIC and Pascal are the two most commonly used languages at both the secondary and introductory college levels, whether in science, math or computer science. In one sense these two and all other programming languages are alike. The hardware of a computer consists of things like registers and memory cells and can only be communicated with via binary codes. Programming languages allow the user to think in terms of variables, data and computations while still communicating with the computer's hardware. On the other hand each language has its own grammar and syntax; and while programs for the same outcome coded in two different languages may have some resemblances, more often than not they are quite distinct. Furthermore language differences extend to power, efficiency and structure. Depending on what one requires in a language these differences will be more or less important.

Technically, Pascal is superior to BASIC as a language for serious, adult programmers (see Bork, 1971 a & 1971 b; Dessy, 1983; Masterson, 1984; and Tesler, 1984). It is generally more efficient

and more powerful, and it inherently encourages the use of good, modular programming technique. In contrast BASIC is often referred to as the "qwerty" typewriter of computer programming languages; that is, it attained widespread use not because it is the best language currently available but because it was the first microcomputer language available. The opposition to BASIC has at times been quite polemical. In fact an article by Mundie (1978) is subtitled "A Polemical Comparison Of [Pascal and BASIC] As General Purpose Microprocessor Languages." Bork refers to BASIC as the "junk food" of programming languages and claims that the "better universities" do not use it (1985, p. 29). Dijkstra says that students exposed to BASIC are "mentally mutilated beyond hope of regeneration" vis-a-vis their potential to become good programmers (1982, p. 14); and Papert claims that BASIC programming requires "... so labyrinthine a structure that ... only the most motivated and brilliant (mathematical) children..." learn to program beyond the level of triviality (1980, p. 35).

Several points of objection can be distilled from the rhetoric. For instance, BASIC lacks many of the advanced programming features found in Pascal and which give to Pascal its superior efficiency and power. Although this is true, one must ask if it really makes any difference at the introductory level. For example, the efficiency and power required for a typical program written by a secondary level chemistry student to do gram-mole conversions is easily accommodated within BASIC. At the school level to say that BASIC is less efficient and powerful than Pascal, conveys about as much information as saying that

microcomputers are less efficient and powerful than mainframe computers.

Although The above issue of advanced programming features is often raised, the gravemen of the case against BASIC vis-a-vis Pascal lies in two quite different points:

1. BASIC is not a structured language like Pascal and using it fosters poor programming habits that are very difficult to break,

2. there is no "ease of learning" advantage that would favor the use of BASIC over Pascal with introductory students.

These are important instructional concerns and if the objections are valid then the use of BASIC should be reconsidered.

The Objection to Non-Structured Programming

Structured computer programs are preplanned, modular, heirarchially arranged, readable programs (see Dijkstra, 1976). They tend to be more efficient and certainly are easier to debug than non-structured programs. BASIC allows structured programming, but the relationship with structure is laissez faire not demanded as with Pascal. Following the path of least resistance, most students approach BASIC programming solutions to a problem with little forethought or planning. They program on a "stream of conscious" basis. As programs lengthen, the lack of planning usually results in more and

more required "GOTO" statements for the proper control of program execution. For lengthy programs this unplanned, profligate use of "GOTO" statements results in spaghetti code, a tangled knot of execution pathways, a nightmare to debug. Students who are accustomed to programming in this manner are the ones Dijkstra calls "mentally mutilated."

The Pascal advantage is that structure is demanded, not merely allowed by the nature of the programming language. The question once again is, does it make any difference at the secondary, introductory level? Obviously people like Dijkstra and Bork think that it does. However their conclusion about the deleterious effects of unstructured, BASIC programming on future programming ability is based on personal observations. There are no empirical studies that implicate BASIC as the causal agent. In fact the hypothesis that learning unstructured programming hinders the later acquisition of structured programming skills has not yet been tested in an experimental setting. The closest studies to this are more general studies of predictors of success in college computer science courses and programs. Factors such as high school math and science background, SAT scores, IQ, and gender have been found to have positive correlations with success in computer science studies which would include structured programming (Alspaugh, 1972; Butcher & Muth, 1985; Campbell & McCabe, 1984; Konvalina et al, 1983). In two of these studies, prior computer programming experience was included as an independent variable but was not found to correlate significantly with college computer science

success (Butcher & Muth, 1985; Konvalina et al, 1983). Although the type of prior programming experience is not specified in these studies, one may safely assume that it included BASIC since this is the most widely taught programming language at the secondary level. Therefore without further evidence one must at least consider the possibility that Dijkstra's observations have a cause other than BASIC.

It also should be noted that not everyone who teaches in the field of programming is a proponent of the structure dogma. There are those that favor BASIC precisely because it does allow the freeform, unstructured type of programming that Pascal prohibits. This is not a lack of planning, but planning in action as opposed to preplanning (Rogoff and Gardner, 1983). Galanter says "BASIC lets you write programs the way that Mozart wrote music, by improvising as you go along" (1983, p. 147). These people recognize that such programming sacrifices efficiency and ease of debugging, but consider the gain in creativity worth the price. For the secondary science teacher, wanting his students to concentrate on science concepts and not on programming technique, this may well be a more useful attribute than structure.

The Problem of Cognitive Matching

BASIC is objected to on the grounds that it has no "ease of learning" advantage over Pascal. No reason exists then for not choosing a superior language. Ease of learning depends on the cognitive match between curriculum demand and cognitive development; the closer

the match, the greater the ease of learning. In this case curriculum demand refers to the cognitive prerequisites necessary for learning BASIC or Pascal. The objection assumes that the minimum cognitive prerequisites for learning both Pascal and BASIC are equally matched to the cognitive development levels of typical secondary students.

The question of cognitive matching has received more attention at the elementary level generally because of Papert's (1980) work with LOGO. Information on using computers with elementary age children frequently notes that a knowledge of child development should guide appropriate computer use (e.g. Clements, 1985, p. 127). Zajonc explains that by inserting "...a device involving formal operations into earlier periods of child development, we risk doing violence to the orderly and natural course of child development" (1984, p. 576). The potential problem is essentially one of cognitive mismatching. The same concern for the relationship between cognitive development and computer use (especially programming) at the secondary level has not been expressed.

It is instructive at this point to consider the British work in science education by Shayer and Adey (1983). They studied the cognitive match between the curriculum demand of the Nuffield Science Programs and the cognitive development levels of the secondary students in those programs. Curriculum demand was determined by a curriculum analysis taxonomy that "arranges and classifies [curriculum] objectives into groups according to (a) the schema or reasoning

patterns employed, and (b) the stage of cognitive development of which they are characteristic" (p. 70). The data on student cognitive development levels came from a study in which 12,000 nine to sixteen year olds were asked to complete a set of tasks comprising the Science Reasoning Tasks Inventory. The results supported Shayer and Adey's main argument, i.e., "...there is a massive mismatch in secondary schools between the expectations institutionalized in courses, textbooks, and examinations and the ability of children to assimilate the experiences they are given" (p. vi).

Given that secondary science curricula, as demonstrated by the Shayer-Adey study, are prone to the problem of cognitive mismatching, it seems eminently reasonable to assume that computer programming languages could also cause the same problem; and if cognitive development and curriculum demand are important factors in selecting a curriculum, then they also should be important factors in selecting a programming language. A comparison of the results of a cognitive demand study of BASIC and Pascal and what is known about the cognitive development levels of typical secondary students would be quite informative. Though there are no studies of this exact nature there are related studies that can supply useful information.

These studies are in the relatively new research field of the applied psychology of the computer user. Within this field the term "task structure" refers to the grammar, syntax and structure of a programming language; and it is the "task structure" that determines the

curriculum demand of a language. The guiding model for research on the applied psychology of computer use is given by Moran (1981, p. 3; also see Newell and Simon 1972):

```
user's goal  +
   task structure  +
      user's knowledge  +
         user's processing limits  =  user's behavior
```

The "user's goal" is simply what the user intends to accomplish by using the computer. A biology student, for example, may wish to have the computer plot the average growth over a series of days of a set of experimentally treated plants. "Task structure" is what will be used to accomplish the user's goal. "User's knowledge" refers to how well the user is able to exploit the task structure. The "user's processing limits" refers to both the user's natural ability and level of cognitive development.

For the purpose of analyzing students' mastery of BASIC programming, the task structure of BASIC can be broken down into six levels:

1. machine
2. transaction
3. prestatement
4. statement
5. chunk
6. program

Level 4, "statement", literally refers to programming statements such as PRINT, LET, READ, etc., and subsumes three more detailed levels of

programming knowledge. Statements in turn are the primary building blocks of programming. Groups of statements such as would form a module are called "chunks", level 5. There are different types of chunks and level 5 itself can be broken down into three levels of complexity. Lastly, one or more chunks combine to form a "program", level 6 (see Mayer, 1979, p. 590).

The difference between an expert programmer and a novice is the expert's ability for greater exploitation of the language task structure. An expert, because he has a good understanding of all structure levels (with the possible exception of level 1, i.e. machine language) is not limited to simple statements, but in fact writes programs based on the upper, more complex levels of the language task structure. The resulting programs fit the Dijktsra definition of "structured." In contrast the novice has a very limited ability to exploit the upper task structure levels and is largely confined to constructing programs statement by statement (Moran, 1981; Mayer, 1981; Sheil, 1981; Shneiderman, 1980).

From a Piagetian perspective the levels of BASIC's task structure can be characterized as either "more concrete" or "more formal" in nature (see Shneiderman, 1985). At the "Statement" level, there is a one-to-one correspondence between most BASIC statements and a particular action that contributes to the concreteness of the concept "Statement". For example:

Statement	Action
10 PRINT "HI"	HI

When statements are grouped together as "chunks" to form structures such as nested loops and recursions, the direct correspondence between statement and action is lost. Thus, most chunks are "more formal" in nature.

In terms of curriculum demand, one would expect less demand at the "statement" level than at the "chunk" level. A student with lower "user processing limits" constructs a program based on statements, while a student with higher "user processing limits" builds a program from chunks. In practice many secondary students learn simple BASIC quite quickly. A reasonable explanatory hypothesis is that the range of curriculum demand within the task structure of BASIC correlates well with the range of cognitive levels typically found amongst secondary students. There is, in other words, good cognitive matching between BASIC task structure and secondary students' cognitive maturity.

Given fundamental, structural simularities among all programming languages, one can assume that the six task structure levels of BASIC can as well be applied to Pascal. The difference between the two languages is a result of restrictions on the use of task structure levels. While BASIC allows the user to build programs from either statements or chunks, Pascal forces the user to mainly work at the upper levels; it demands structure. This is in fact the major Pascal "advantage" according to Pascal proponents. Mundie is quite explicit,

"Pascal...allows the programmer to remain on a high level of algorithmic abstraction, where he functions best..." (1978, p. 42). An adult, expert programmer may indeed function best at "...a high level of abstraction ...", but the same can hardly be said for the typical adolescent. What is an advantage for the mature programmer is quite likely to be a substantial disadvantage for secondary students. Some corroboration of this is found in a 1979 study by LaFrance. He attempted to teach the concept of structured Pascal programming via a game-format to a group of gifted nine to twelve year olds but was generally unsuccessful. The fact that these were "gifted" nine to twelve year olds lends credence to the hypothesis that a similar study with secondary students would have the same result.

Conclusion

If secondary science teachers are not going to employ computer programming as an instructional technique with any greater frequency than is now done, then the foregoing is only so much theoretical arcana. However, as stated at the beginning, the increase in computer literacy requirements and the increase in the number of personally owned computers gives rise to the distinct prospect of increased use of student programming in the science classroom. Accepting that, one must face two competing, yet legitimate concerns. Computer scientists are concerned about the prospect of students acquiring improper programming habits, habits that will ill serve them if they later choose

to enter computer related studies. On the other hand instructional experts are more concerned that instructional methods be well suited to students' levels of cognitive maturity and to the subject being taught.

For many reasons, not the least of which is financial, it can be expected that BASIC will remain the language of the status quo for some time to come. At this point the widespread use of BASIC should not be considered cause for alarm since the evidence that is currently available suggests that BASIC rather than Pascal, is cognitively better suited for use with secondary level students. Furthermore there is as of now no empirical evidence that prior knowledge of unstructured programming has a significant negative influence on the acquisition of structured programming skills. It would be wiser to avoid rhetorical arguments about the merits and demerits of BASIC and Pascal and instead pursue relevant research topics, i.e. a thorough examination of the roles that programming can play in science education, a rigorous analysis of the task structure and curriculum demand of various languages especially Pascal, and a controlled study of the effect of prior unstructured programming experience on the acquisition of structured programming skills.

References

Abernathy, S. M. & Pettibone, T. J. (1984). Computer literacy and certification: What States are doing. T.H.E. Journal, 12(4), 117-120.

Alspaugh, C. A. (1972). Identification of some components of computer programming aptitude. Journal for Research in Mathematics Education, 3, 89-98.

Bayman, P. (1983). The effects of instructional procedures on beginning programmers' mental models. ERIC document No. 238 406.

Bayman, P. & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. Communications of the ACM, 26(9), 677-679.

Bork, A. (1971 a). Science teaching and computer languages. ERIC document No. 060 626.

Bork, A. (1971 b). Introduction to computer programming languages. ERIC document No. 060 620.

Bork, A. (1985). Personal computers for education. New York: Harper & Row.

Butcher, D. F. & Muth, W. A. (1985). Predicting performance in an introductory computer science course, 28(3), 263-268.

Campbell, P. F. & McCabe, G. P. (1984). Predicting the success of freshmen in a computer science major. Communications of the ACM, 27(11), 1108-1113.

Clements, D. H. (1985). Computers in early childhood education. Educational Horizons, 63(5), 124-127.

Dijkstra, E. W. (1976). A discipline of programming. Englewood Cliffs: Prentice-Hall.

Dijkstra, E. W. (1982). How do we tell truths that might hurt? Sigplan Notices, 17(5).

Dessy, R. E. (Ed.) (1983). Languages for the laboratory. Analytical Chemistry, 55(6), 650A-766A.

Galanter, E. (1983). Kids and computers. New York: The Putnam Publishing Group.

Konvalina, J. et al (1983). Math proficiency: a key to success for computer science. Communications of the ACM, 26(5), 377-382.

LaFrance, J. E. (1979). Shall we teach structured programming to children? ERIC document No. 192 767.

Masterson, F. A. (1984). Languages for students. BYTE, 9(6), 233-234,236,238.

Mayer, R. E. (1979). A psychology of learning BASIC. Communications of the ACM, 22(11), 589-593.

Mayer, R. E. (1981). The psychology of how novices learn computer programming, ACM Computing Surveys, 13(1), 121-141.

Moran, T. P. (1981). An applied psychology of the user. ACM Computing Surveys, 13(1), 1-11.

Moyer, P. C. (1985). Structured programming techniques in BASIC. Journal of Computers in Mathematics and Science Teaching, 4(2), 60-62.

Mundie, D. A. (1978). Pascal vs BASIC: a polemical comparison of the two as general purpose microprocessor languages. People's Computers, 6(4), 41-47.

Newell, A. & Simon, H. A. (1972). Human problem solving. Englewood Cliffs: Prentice-Hall.

Office of Technology Assessment (1982). Information technology and it impact on American education. Washington, D.C.

Papert. S. (1980). Mindstorms. New York: Basic Books.

Rogoff, B. & Gardner, W. P. (1983). Guidance in cognitive development: an examination of mother-infant instruction. In B. Rogoff & J. Lave (Eds.), Everyday cognition: Its development in social context. Cambridge: Harvard University Press.

Shavelson, R. J. et al (1984). Teaching mathematics and science: patterns of microcomputer use. The RAND Corporation, R-3180-NIE/RC.

Shayer, M. & Adey, P. (1983). Towards a science of science teaching. London: Heinemann Educational Books. See also Pea, R. D. & Kurland, D. M. (1983). On cognitive prerequisites of learning computer programming (Technical Report No. 18). New York: Bank Street College of Education.

Shell, B. (1981). Psychological study of programming. ACM Computing Surveys, 13(1), 101-120.

Shneiderman, B. (1980). Software psychology: Human factors in computer and information systems. New York: Winthrop Co.

Shneiderman, B. (1985). When children learn programming: antecedents, concepts and outcomes. The Computing Teacher, 12(5), 14-17.

Tesler, L. G. (1984). Programming languages. Scientific America, 251(3), 70-78.

Zajonc, A. G. (1984). Computer pedagogy? Questions concerning the new educational technology. Teachers College Record, 85(4), 569-577.